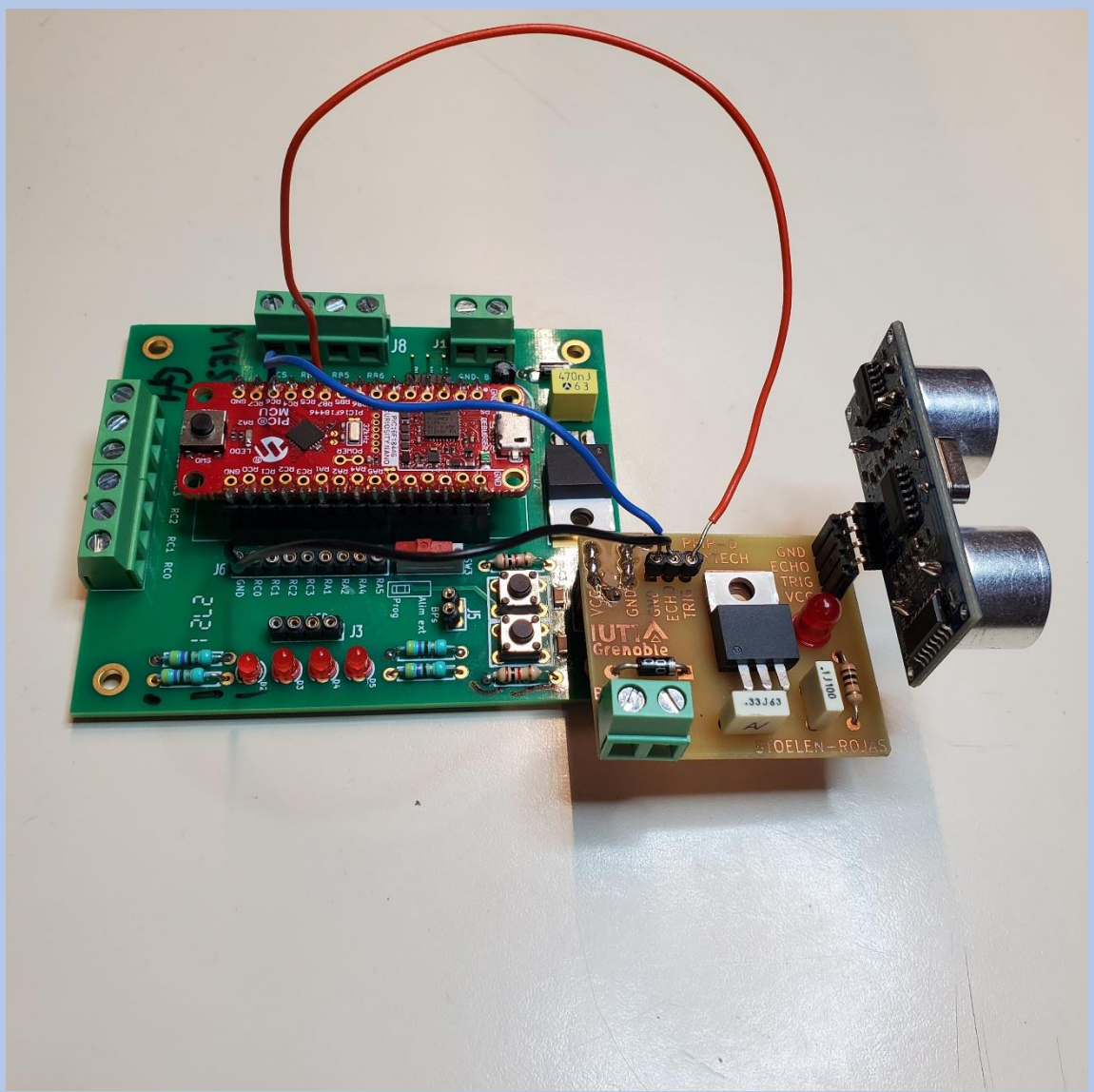
Compte-Rendu Programmation Radar



*Ce n’est pas notre carte avec les borniers à l’envers

Table des matières

Présentation du projet et de son cahier des charges

L'objectif du projet est de programmer le PIC16F18446 pour faire fonctionner trois capteurs à ultrasons HC SR04 nous permettant d'avoir un retour sur la distance mesurée des trois capteurs sur un terminal.

Nos trois capteurs seront répartis de sorte à en avoir un à gauche, un au milieu et un à droite.

Le cahier des charges

- Se servir de trois capteurs à ultrasons HC SR04.

- Fréquence de mesure des capteurs à 3.3Hz chacun.

- Se servir du SMT pour la mesure de la distance mesurée par les capteurs.

- Affichage des distances sur terminal en se servant de la liaison série de l'EUSART.

- Rapidité de 115,200 bauds pour la liaison série

- 8 bits de données avec un bit de Stop et un bit de Start sans parité.

- Broches imposés :

Signal	Nature (entrée ou sortie)	Broche utilisé sur le PIC
Commande du capteur gauche	Sortie digital	RC0
Commande du capteur central	Sortie digital	RC1
Commande du capteur droit	Sortie digital	RC5
Impulsion de largeur variable capteur gauche	Entrée digital	RC4
Impulsion de largeur variable capteur central	Entrée digital	RC6
Impulsion de largeur variable capteur droit	Entrée digital	RC7
Communication TxD	Sortie digital	RB4
Communication RxD	Entrée réception	RB6

Gestion de d'envoi du signal pour les trois capteurs

Nous devrons lire l'information des trois capteurs sur le terminal avec une fréquence de 3.3Hz chacun.

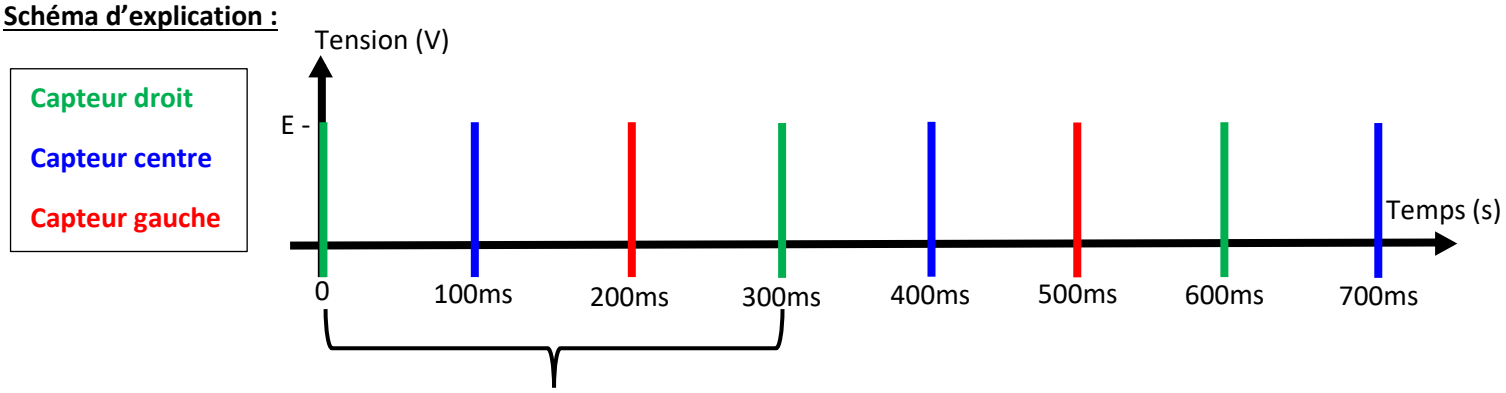
Nos capteurs à ultrasons HC SR04 fonctionnent de telle sorte à ce que lorsque l'on leur envoie une impulsion de plus de 10µs sur leur broche TRIG, ils nous renvoient après avoir fini leur mesure, une impulsion d'une durée qui varie en fonction de la distance sur la broche ECHO. Leur distance maximale est de 4m et leur précision est de 3mm.

Donc si l'on veut mesurer la durée de l'impulsion de sortie sur ECHO, nous devons compter le temps qui s'écoule entre le front montant de l'impulsion et son front descendant. Nous devons donc nous servir d'un périphérique spécialisé dans le comptage d'une durée d'impulsion propre au PIC16F18446. Ce périphérique se nomme le SMT (Signal Measurement Timer).

Notre microcontrôleur ne contient qu'un seul périphérique SMT or, nous avons trois capteurs à ultrasons à contrôler à une fréquence de 3.3Hz chacun. Ce qui implique que nous ne pouvons pas contrôler ces capteurs en même temps, nous devons le faire les uns après les autres à la suite. Ce qui serait donc possible, c'est d'alterner leur commande, dans un premier temps on commande le capteur de droite, juste après celui du centre et après celui de gauche et rebelote on recommence.

Étant donné que la fréquence est de 3.3Hz, cela veut dire que nous ne pourrons commander un capteur que toute les 303ms environ. Comme nous avons trois capteurs à contrôler, cela revient à contrôler un capteur toute les 100ms. Cela permettrait de garder la fréquence imposée de 3.3Hz environ.

Schéma d'explication :



On garde toujours une période de 300ms soit ~3.3Hz

Pour avoir une période pour les impulsions toute les 100ms, il nous faut avoir une fréquence de 10Hz. On utilise le Timer0 sur 16bits.

La durée d'impulsion est directement liée au temps qu'il a fallu au son émis pas le capteur à ultrasons de parcourir une distance puis de rebondir sur l'objet pour revenir au capteur. Exemple, si la durée d'impulsion est de 20ms, cela signifie que la durée nécessaire au son pour atteindre l'objet a été de 10ms.

Calcul de la vitesse du son

Maintenant que nous avons une durée il nous faut la convertir en une distance :

$$Vitesse\ du\ son = 20.05 \times \sqrt{température\ en\ Kelvin}$$

Ici la température pour cet exemple est de 25°C soit 298.15°K

Donc notre vitesse du son est de $20.05 \times \sqrt{298.15} \approx 346 m.s^{-1}$

Avec cet exemple, 10ms correspond à une distance de $d = 346 \times 0.01 = 3.46m$

Mesure théorique maximale de la distance que peut calculer le SMT

Cependant comment cette durée d'impulsion est mesurée par notre microcontrôleur ? Le périphérique SMT fonctionne de telle sorte à ce que lorsqu'il détecte un front montant, il va se mettre à compter très rapidement jusqu'à capter le front descendant. Pour avoir une mesure du temps la plus précise possible, on va se servir de l'horloge interne la plus rapide du PIC16F18446, soit une horloge de 32Mhz. Sachant que le registre du SMT qui va stocker cette valeur de comptage de la durée d'impulsion est un registre de 24bits.

Cela signifie que la distance maximale mesurable avec 32MHz est de :

$$2^{24} \times \frac{1}{32MHz} = 524ms \qquad \frac{524ms \times 346}{2} \approx 91m$$

Donc notre mesure maximum est de 90m ce qui est largement suffisant comparé à notre capteur qui ne peut mesurer que jusqu'à 4m.

Mesure maximale de la durée d'envoi de l'EUSART pour notre configuration

Maintenant que nous avons notre valeur de comptage de la durée d'impulsion sur 24bits, il nous faut la transformer dans notre programme pour l'avoir en mm, car l'étape suivante est de l'envoyer à notre terminal grâce à une liaison série géré avec le périphérique spécialisé dans cette tâche du PIC16F18446. Ce périphérique s'appelle l'EUSART cependant, nous avons la contrainte d'envoyer au terminal la distance des trois capteurs en même temps toutes les 300ms.

En résumé nous faisons une mesure des capteurs les uns après les autres toutes les 100ms et nous envoyons leur résultat au terminal toute les 300ms les trois en même temps.

Nous devons alors avoir un temps d'envoi inférieur à 300ms.

On a 24 caractères de 8bits avec leur bit de start et de stop donc 10bits. Nous envoyons trois mesures car trois capteurs à une vitesse de 115,200 bauds. Donc :

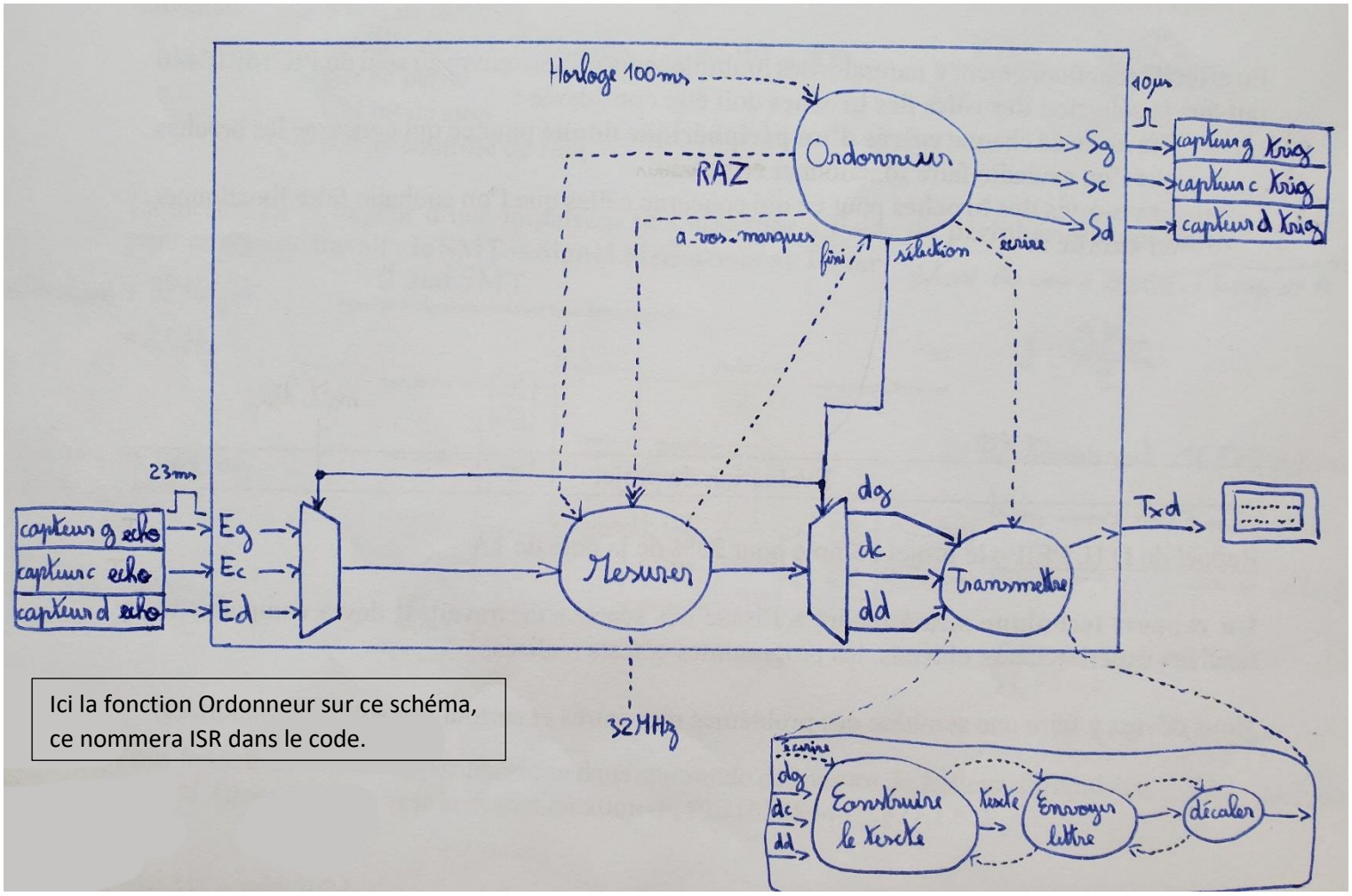
$$durée\ d'envoi = \frac{24 \times 10 \times 3}{115\ 200} = 6.25ms$$

Nous avons alors le temps !! car $6.25ms \ll 300ms$

Fonctionnement global du code

Après avoir fait la réflexion préalable ci-dessus ;

Voici un schéma explicatif du fonctionnement de notre code :



La configuration des entrées sorties

```
114   void E_S (void)
115   {
116    TRISC= TRISC & 0xDC;  //0b1101 1100
117    //RC5 en sortie Sd (trig droite)
118    //RC0 en sortie Sg (trig gauche)
119    //RC1 en sortie Sc (trig centre)
120    TRISC= TRISC | 0xD0;  //0b1101 0000
121    //RC4 en entee (echo gauche)
122    //RC6 en entee (echo centre)
123    //RC7 en entee (echo droite)
124    TRISB= TRISB & 0xEF;  //0b1110 1111
125    //RB4 en sortie (Tx)
126    TRISB= TRISB | 0x40;  //0b0100 0000
127    //RB6 entree (Rx)
128    ANSELB=0x00; //port B en TOR
129    ANSELC=0x00; //port C en TOR
130   }
```

La configuration des interruptions

```
207  void confg_interruption(void)
208  {
209    TMR0IE=1; // autorisation des interruptions liees au timer0
210    SMT1PWAIE=1;  //autorisation des interruptions SMT1 pour la mesure de largeur d'impulsion (Pulse Width Acquisition)
211    PEIE=1; // autorisation des interruptions périphériques
212    GIE=1; // autorisation des interruptions globales
213  }
```

La configuration du Timer0 à 10Hz

```
153  void config_timer0(void)// 10HZ
154  {
155    // Configuration du registre T0CON0
156    // 0x90 = 0b1001 0000
157    // 1 (bit 7) : Timer0 activé
158    // 0 (bit 6) : Non utilisé (toujours à 0)
159    // 1 (bit 5) : Mode 16 bits
160    // 0000 (bits 4-0) : Post-diviseur à 1:1
161    T0CON0 = 0x90;
162
163    // Configuration du registre T0CON1
164    // 0x54 = 0b0101 0100
165    // 010 (bits 7-5) : Source d'horloge Fosc/4
166    // 1 (bit 4) : Synchronisation désactivée
167    // 0100 (bits 3-0) : Pré-diviseur à 1:16
168    T0CON1 = 0x54;
169
170    // Initialisation du registre TMR0 pour générer l'interruption à 10Hz
171    // Calcul du préchargement : 65536 - (50000) = 15536
172    // En hexadécimal : 15536 = 0x3CB0
173    TMR0H = 0x3C; // Partie haute du registre TMR0 (0x3C)
174    TMR0L = 0xB0; // Partie basse du registre TMR0 (0xB0)
175
176    // Commentaire sur le calcul de la fréquence
177    // Fréquence du Timer = Fosc / (Pré-diviseur * Post-diviseur * (65536 - TMR0))
178    // 32 MHz / (4*16 * 50000) = 10 Hz
179  }
```

La configuration du SMT (Signal Measurement Timer)

```
133  void config_SMT(void)
134  {
135      // Configuration du registre SMT1CON0
136      SMT1CON0 = 0x80; // 0b1000 0000
137      /*
138      Bits de SMT1CON0 :
139      1 : SMT enable (activation du SMT)
140      x : Non utilisé (toujours à 0)
141      0 : Le compteur se réinitialise à l'interruption SMT1PR
142      0x : La fenêtre se déclenche sur le front montant
143      0 : SMT actif-haut / front montant activé
144      0 : SMT incrémente sur le front montant du signal d'horloge sélectionné
145      00 : Diviseur de fréquence (prescaler) 1:1
146      */
147
148      // Configuration du registre SMT1CON1
149      SMT1CON1 = 0x01; // 0b0000 0001
150      /*
151      Bits de SMT1CON1 :
152      0 : GO (Incrémentation et acquisition de données désactivées)
153      0 : single acquisition
154      x : Non utilisé (toujours à 0)
155      x : Non utilisé (toujours à 0)
156      0001 : gated timer mode
157      */
158
159      // Configuration du registre SMT1CLK
160      SMT1CLK = 0x01; // 0b0000 0001
161      /*
162      Bits de SMT1CLK :
163      xxxxx : Non utilisé (toujours à 0)
164      001 : Source d'horloge Fosc
165      */
166
167      // Configuration du registre SMT1SIG
168      SMT1SIG = 0x00; // Pin sélectionnée par SMT1SIGPPS p328
169      /*
170      Bits de SMT1SIG :
171      Tout à 0 : Utilise la broche sélectionnée par SMT1SIGPPS
172      */
173  }
```

La configuration de la liaison série

```
201  void init_serie(void)
202  {
203      SP1BRG=16; //  p413
204      /*CSRC : '0' sans importance car on est en mode asynchrone
205      TX9 : '0' transmission de 8 bits
206      TXEN : '1' Autorisation de la transmission
207      SYNC : '0' Mode de fonctionnement asynchrone5 (8 bits )
208      SENDB : '0'
209      BRGH : '1' (8 bits )
210      TRMT : '0' c'est un bit en lecture seule, donc sans importance !
211      TX9D : '0' le neuvi me bit sert de bit de STOP*/
212      TX1STA=0x24;//0b 0010 0100
213      /*SPEN : '1' validation de l'affectation au port s rie de RC6 et RC7
214      RX9 : '0' r ception de 8 bits
215      SREN : '0' Sans importance pour le mode asynchrone
216      CREN : '1' Validation de la r ception
217      ADDEN : '0' invalidation de la d tection d'adresse
218      FERR : '0' c'est un bit en lecture seule, donc sans importance !?
219      OERR : '0' c'est un bit en lecture seule, donc sans importance !
220      RX9D : '0' c'est un bit en lecture seule, donc sans importance !*/
221      RC1STA=0x90; //1000 0000
222      BAUD1CON = 0x00 ;//0b xxxx 0(8 BRG16=0)xxx
223      RB6PPS=0x0E;//xxx 01 110 rb6 on cable notre RX qui vient de CDCTX
224      RB4PPS=0x0F;//xx 00 1111 rb4 on cable notre TX qui va sur CDCRX p231(pour RB4PPS ou CK1PPS) et page 233 pour la valeur 00 1111
225  }
```

La fonction transmettre

```
235  void transmettre(unsigned int mesure,unsigned char tab[])
236  {
237      short int i;
238      // Extraction des milliers, centaines, dizaines et unités de la mesure (mesure en mm)
239      unsigned int m = mesure/1000;
240      unsigned int dm = mesure%1000/100;
241      unsigned int cm = mesure%100/10;
242      unsigned int mm = mesure%10;
243      // Conversion des valeurs numériques en caractères ASCII et stockage dans le tableau
244      tab[16] = (m + '0');
245      tab[17] = (dm + '0');
246      tab[18] = (cm + '0');
247      tab[20] = (mm + '0');
248
249      // Envoi de la chaine de caracteres
250          for (i = 0; tab[i] != '\0'; i++)
251          {
252              TX1REG = tab[i];  // Envoi du caractere contenu dans le tab a la position i
253              while (TRMT==0); // Attente de la fin de la transmission
254          }
255  }
```

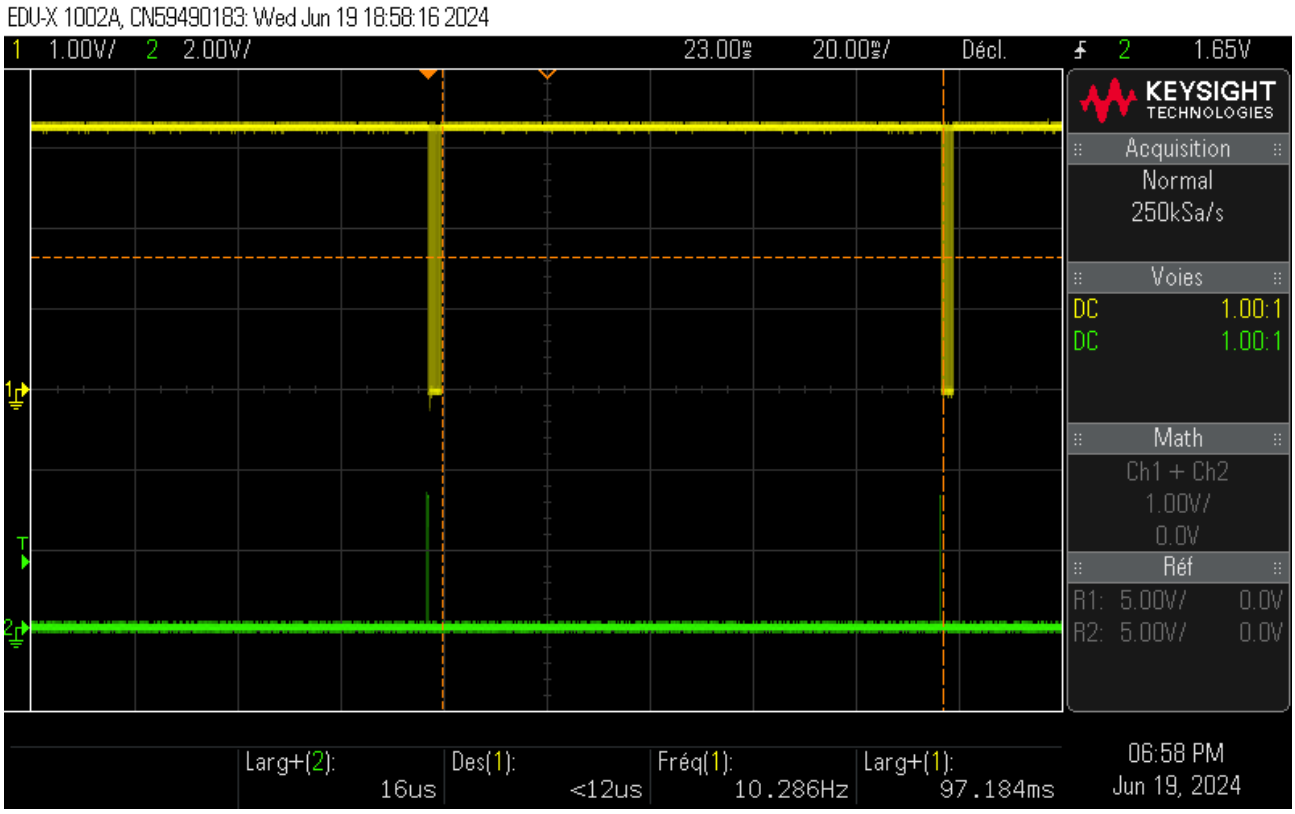
Fonctionnement de L'ISR

```
void __interrupt() ISR ( void )
{
    static short int selec_capteur = 1;
    if ( TMR0IF ) // SI overflow du timer 0 (fin de comptage des 100ms))
    {
        // Réinitialiser le Timer0 pour la prochaine période de comptage
        TMR0H = 0x3C; // Partie haute du registre TMR0 (0x3C)
        TMR0L = 0xB0; // Partie basse du registre TMR0 (0xB0)
        TMR0IF = 0;   // RAZ du flag TMR0IF

        // Réinitialiser le Timer SMT1
        SMT1TMR = 0; // RAZ du timer SMT1
        SMT1CON1=SMT1CON1|0x80; //ob1000 0000 Déclencher le SMT1 (GO)

        // Sélectionner le capteur et générer une impulsion
        switch (selec_capteur)
        {
            case 1:
                SMT1SIGPPS=0x17; //selection de la broche RC7 0bxxx 10 111 en entree du SMT1 (trig capteur droite) p239
                //generation d une impulsion sur la broche echo du capteur droite
                RC5=1;
                __delay_us(15);
                RC5=0;
                break;
            case 2:
                SMT1SIGPPS=0x16; //selection de la broche RC6 0b0xxx 10 110 en entree du SMT1 (trig capteur centre) p239
                //generation d une impulsion sur la broche echo du capteur centre
                RC1=1;
                __delay_us(15);
                RC1=0;
                break;
            case 3:
                SMT1SIGPPS=0x14; //selection de la broche RC4 0b0xxx 10 100 en entree du SMT1 (trig capteur gauche) p239
                //generation d une impulsion sur la broche echo du capteur gauche
                RC0=1;
                __delay_us(15);
                RC0=0;
                break;
            default :
                selec_capteur=1;
                break;

        }
    }
    if ( SMT1PWAIF )// Vérifier si le drapeau d'interruption de largeur d'impulsion SMT1 est levé
    {
      SMT1PWAIF=0;  // RAZ du flag d'interruption de largeur d'impulsion SMT1
      unsigned long int mesure=0;
      mesure=SMT1CPW; // Lire la mesure du SMT1
      // Calculer la distance (mm) en utilisant la mesure
      mesure=(mesure*(1/32000000.0)*Vitesse_son/2.0*1000.0);
      // Stocker la mesure en fonction du capteur sélectionné
      switch (selec_capteur)
        {
            case 1:
                mesure_d=mesure; // Stocker la mesure pour le capteur droite
                selec_capteur++;
                //ecrire=1;
                break;
            case 2:
                mesure_c=mesure; // Stocker la mesure pour le capteur centre
                selec_capteur++;
                break;
            case 3:
                mesure_g=mesure; // Stocker la mesure pour le capteur gauche
                selec_capteur=1;
                ecrire=1; // Déclencher l'envoi des mesures
                break;
            default :
                selec_capteur=1;
                break;
        }
    }
}
```

Tache de fond

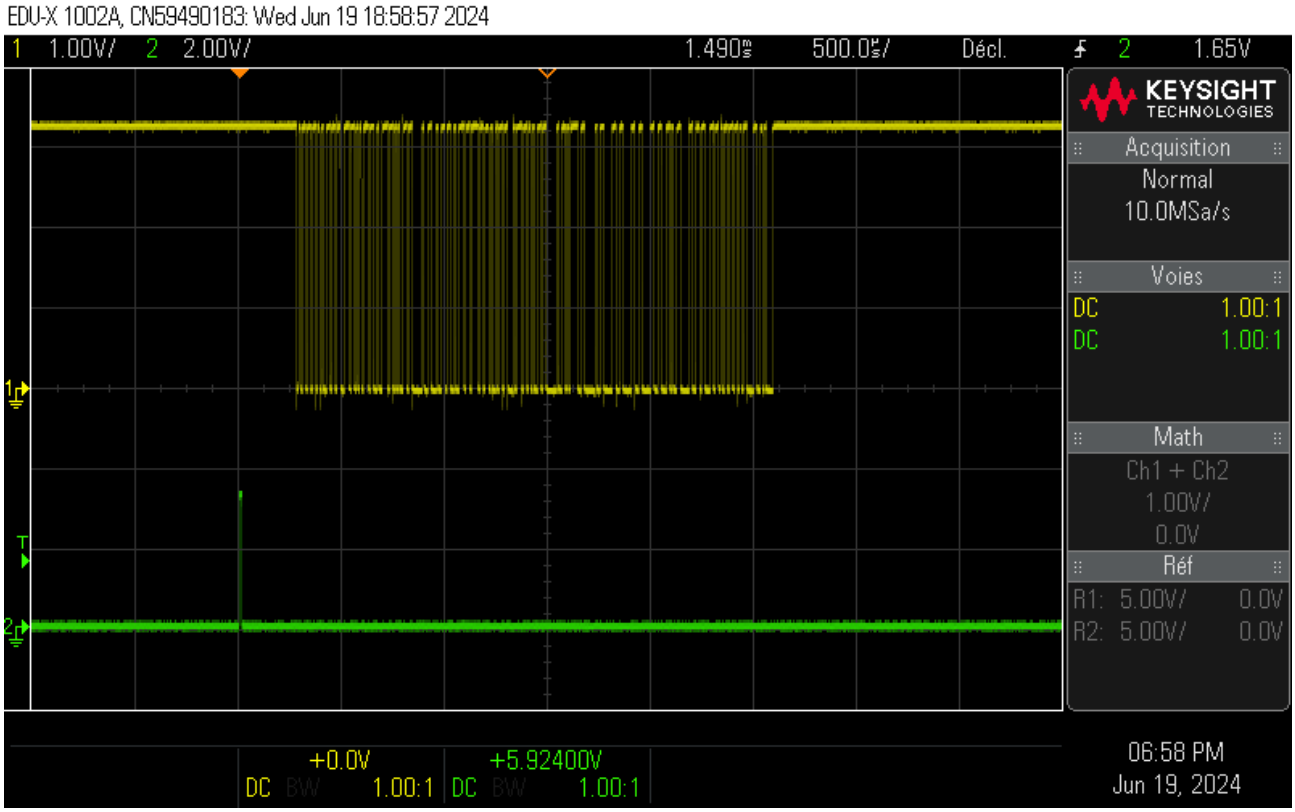
```c
#include "config_bit.h"  // Inclusion des bits de configuration
#define _XTAL_FREQ 32000000 // Fréquence de l'oscillateur
#define Vitesse_son 345 // Vitesse du son en m/s
//PPS p230
// Prototypes des fonctions
void E_S(void);
void config_SMT(void);
void config_timer0(void); // Configuration du Timer0 pour 10Hz
void confg_interruption(void); // Configuration et autorisation des interruptions
void init_serie(void);
void transmettre(unsigned int mesure, unsigned char tab[]);
void __interrupt() ISR(void); // Routine d'interruption

// Variables globales
short int ecrire = 0; // Indicateur d'écriture
unsigned int mesure_d = 0; // Mesure du capteur de droite
unsigned int mesure_c = 0; // Mesure du capteur central
unsigned int mesure_g = 0; // Mesure du capteur de gauche

void main(void)
{
  E_S();
  config_timer0();
  init_serie();
  config_SMT();
  confg_interruption(); //configuration et autorisation des interruptions
  //                        0         1         2
  //                        012345678901234567890123 456789012345
  unsigned char tab_d[] = {"mesure droite = 000,0cm\r\n"};
  //                        0         1         2
  //                        012345678901234567890123 456789012345
  unsigned char tab_c[] = {"mesure centre = 000,0cm\r\n"};
  //                        0         1         2
  //                        012345678901234567890123 456789012345
  unsigned char tab_g[] = {"mesure gauche = 000,0cm\r\n"};
  while(1==1)
  {
    if(ecrire)
    {
      ecrire=0;
      transmettre(mesure_g,tab_g);
      transmettre(mesure_c,tab_c);
      transmettre(mesure_d,tab_d);
    }
  }

}
```

Fonctionnement avec oscilloscope



En vert nous avons l'impulsion de déclenchement sur la broche TRIG du capteur à ultrasons.

En jaune c'est l'envoie par liaison série des caractères en ASCII sur la broche RB4 du microcontrôleur indiquant la distance mesurée venant de l'EUSART.



Conclusion du projet avec son fonctionnement

En conclusion, lors de ce projet de codage, Nous nous sommes servis des différents périphériques mit à notre disposition telle que :

- L'EUSART
- Le SMT
- Le TIMER 0

Nous avons dû aussi nous servir des interruptions et de la configuration des entrées sorties.

https://github.com/MathisRojas/Radar-ultrason

(Lien pour consulter le programme entier en ligne)

Nous n'avons pas pu tester notre code avec trois capteurs, car nous n'en avons qu'un seul par groupe. Mais le code fonctionnera avec les trois capteurs.

Quant à son fonctionnement, nous l'avons testé sur toutes les longueurs possibles proches comme loin. À partir de 3m80, la mesure de la distance n'est plus fiable, et au-delà de 4m, la mesure est erronée et aléatoire.